COT 6410 Fall 2010 Final Exam Sample Questions

1. Let set **A** be recursive, **B** be re non-recursive and **C** be non-re. Choosing from among **(REC) recursive**, **(RE) re non-recursive**, **(NR) non-re**, categorize the set **D** in each of a) through d) by listing **all** possible categories. No justification is required.

   a.) D = ~C RE, NR

   b.) D ⊆ (A∪C) REC, RE, NR

   c.) D = ~B NR

   d.) D = B – A REC, RE

2. Prove that the **Halting Problem** (the set K_0) is not recursive (decidable) within any formal model of computation. (Hint: A diagonalization proof is required here.)

   Assume we can decide the halting problem. Then there exists some total function Halt such that

   $$\text{Halt}(x,y) = \begin{cases} 1 & \text{if } [x](y) \text{ is defined} \\ 0 & \text{if } [x](y) \text{ is not defined} \end{cases}$$

   Here, we have numbered all programs and [x] refers to the x-th program in this ordering. We can view Halt as a mapping from ℵ into ℵ by treating its input as a single number representing the pairing of two numbers via the one-one onto function

   $\text{pair}(x,y) = \langle x,y \rangle = 2^x (2y + 1) - 1$

   with inverses

   $\langle z \rangle_1 = \exp(z+1,1)$

   $\langle z \rangle_2 = (((z + 1) // 2^{\langle z \rangle_1}) - 1) // 2$

   Now if Halt exist, then so does Disagree, where

   $$\text{Disagree}(x) = \begin{cases} 0 & \text{if Halt}(x,x) = 0, \text{ i.e, if } [x](x) \text{ is not defined} \\ \mu y\,(y == y+1) & \text{if Halt}(x,x) = 1, \text{ i.e, if } [x](x) \text{ is defined} \end{cases}$$

   Since Disagree is a program from ℵ into ℵ , Disagree can be reasoned about by Halt. Let d be such that Disagree = [d], then

   Disagree(d) is defined ⇔ Halt(d,d) = 0 ⇔ [d](d) is undefined ⇔ Disagree(d) is undefined

   But this means that Disagree contradicts its own existence. Since every step we took was constructive, except for the original assumption, we must presume that the original assumption was in error. Thus, the Halting Problem is not solvable.

3. Using reduction from the known undecidable **HasZero, HZ = { f | ∃x f(x) = 0 }**, show the non-recursiveness (undecidability) of the problem to decide if an arbitrary primitive recursive function **g** has the property **IsZero, Z = { f | ∀x f(x) = 0 }**.

   HZ = { f | ∃x ∃t [STP(x, f, t) & VALUE(x, f, t) == 0] }
   Let f be the index of an arbitrary effective procedure.
   Define $g_f(y) = 1 - \exists x \exists t$ [STP(x, f, t) & VALUE(x, f, t) == 0]
   If ∃x f(x) = 0, we will find the x and the run-time t, and so we will return 0 (1 – 1)
   If ∀x f(x) ≠ 0, then we will diverge in the search process and never return a value.
   Thus, f ∈ HZ iff g_f ∈ Z = { f | ∀x f(x) = 0 }.

4. Choosing from among **(D) decidable**, **(U) undecidable**, **(?) unknown**, categorize each of the following decision problems. No proofs are required.

Problem / Language Class	**Regular**	**Context Free**	**Context Sensitive**
$L = \Sigma^*$?	***D***	***U***	***U***
$L = \phi$?	***D***	***D***	***U***
$L = L^2$?	***D***	***U***	***U***
$x \in L^2$, for arbitrary x ?	***D***	***D***	***D***

5. Use **PCP** to show the undecidability of the problem to determine if the intersection of two context free languages is non-empty. That is, show how to create two grammars G_A and G_B based on some instance $P = \langle\langle x_1,x_2,\ldots,x_n\rangle, \langle y_1,y_2,\ldots,y_n\rangle\rangle$ of **PCP**, such that $L(G_A) \cap L(G_B) \neq \phi$ iff **P** has a solution. Assume that **P** is over the alphabet Σ.You should discuss what languages your grammars produce and why this is relevant, but no formal proof is required.

$G_A = (\{ A \} , \Sigma \cup \{ [i] \mid 1 \le i \le n \} , A , P_A \}$ **$G_B = (\{ B \} , \Sigma \cup \{ [i] \mid 1 \le i \le n \} , B , P_B \}$**

$P_A : A \rightarrow x_i A [i] \mid x_i [i]$ **$P_B : A \rightarrow y_i B [i] \mid y_i [i]$**

$L(G_A) = \{ x_{i_1} x_{i_2} \ldots x_{i_p} [i_p] \ldots [i_2] [i_1] \mid p \ge 1, 1 \le i_t \le n, 1 \le t \le p \}$

$L(G_B) = \{ y_{j_1} y_{j_2} \ldots y_{j_q} [j_q] \ldots [j_2] [j_1] \mid q \ge 1, 1 \le j_u \le n, 1 \le u \le q \}$

$L(G_A) \cap L(G_B) = \{ w [k_r] \ldots [k_2] [k_1] \mid r \ge 1, 1 \le k_v \le n, 1 \le v \le r \}$, where

$$w = x_{k_1} x_{k_2} \ldots x_{k_r} = y_{k_1} y_{k_2} \ldots y_{k_r}$$

If $L(G_A) \cap L(G_B) \neq \phi$ then such a w exists and thus $k_1 , k_2 , \ldots , k_r$ is a solution to this instance of PCP. This shows that a decision procedure for the non-emptiness of the intersection of CFLs implies a decision procedure for PCP, which we have already shown is undecidable. Hence, the non-emptiness of the intersection of CFLs is undecidable. Q.E.D.

6. Consider the set of indices **CONSTANT** = $\{ f \mid \exists K \forall y [\varphi_f(y) = K] \}$. Use Rice's Theorem to show that **CONSTANT** is not recursive. Hint: There are two properties that must be demonstrated.

First, show CONSTANT is non-trivial.

Z(x) = 0, which can be implemented as the TM R, is in CONSTANT
S(x) = x+1, which can be implemented by the TM $C_1$1R, is not in CONSTANT
Thus, CONSTANT is non-trivial

Second, let f, g be two arbitrary computable functions with the same I/O behavior.
That is, $\forall x$, if f(x) is defined, then f(x) = g(x); otherwise both diverge, i.e., f(x)↑ and g(x)↑
Now, $f \in$ CONSTANT
$\Leftrightarrow \exists K \forall x [f(x) = K]$ by definition of CONSTANT
$\Leftrightarrow \forall x [g(x) = C]$ where C is the instance of K above, since $\forall x [f(x) = g(x)]$
$\Leftrightarrow \exists K \forall x [g(x) = K]$ from above
$\Leftrightarrow g \in$ CONSTANT by definition of CONSTANT

Since CONSTANT meets both conditions of Rice's Theorem, it is undecidable. Q.E.D.

7. Show that **CONSTANT** $\equiv_m$ **TOT**, where **TOT** = { **f** | $\forall y\ \varphi_f(y)\downarrow$ }.

CONSTANT $\leq_m$ **TOT**
Let f be an arbitrary effective procedure.
Define g_f by
$g_f(0) = f(0)$
$g_f(y+1) = f(y+1) + \mu z\ [f(y+1) = f(y)]$
Now, if f ∈ CONSTANT then $\forall y$ [f(y)↓ and [f(y+1) = f(y)]].
Under this circumstance, μz [f(y+1) = f(y)] is 0 for all y and $g_f(y) = f(y)$ for all y.
Clearly, then $g_f \in$ TOT
If, however, f ∉ CONSTANT then $\exists y$ [f(y+1) ≠ f(y)] and thus, $\exists y$ f(y)↑.
Choose the least y meeting this condition.
If f(y)↑ then $g_f(y)$↑ since f(y) is in $g_f(y)$'s definition (the 1st term).
If f(y)↓ but [f(y+1) ≠ f(y)] then $g_f(y)$↑ since μz [f(y+1) = f(y)]↑ (the 2nd term).
Clearly, then $g_f \notin$ TOT
Combining these, f ∈ CONSTANT ⇔ $g_f \in$ TOT and thus CONSTANT $\leq_m$ TOT

TOT $\leq_m$ **CONSTANT**
Let f be an an arbitrary effective procedure.
Define g_f by
$g_f(y) = f(y) - f(y)$
Now, if f ∈ TOT then $\forall y$ [f(y)↓] and thus $\forall y\ g_f(y) = 0$. Clearly, then $g_f \in$ CONSTANT
If, however, f ∉ TOT then $\exists y$ [f(y)↑] and thus, $\exists y$ [$g_f(y)$↑]. Clearly , then $g_f \notin$ CONSTANT
Combining these, f ∈ TOT ⇔ $g_f \in$ CONSTANT and thus TOT $\leq_m$ CONSTANT

Hence, CONSTANT $\equiv_m$ **TOT. Q.E.D.**

8. Why does Rice's Theorem have nothing to say about each of the following? Explain by showing some condition of Rice's Theorem that is not met by the stated property.
***a.*) AT-LEAST-LINEAR = { f | $\forall y$ $\varphi_f(y)$ converges in no fewer than y steps }.**
We can deny the 2nd condition of Rice's Theorem since
Z, where Z(x) = 0, implemented by the TM R converges in one step no matter what x is and hence is not in AT-LEAST-LINEAR

Z', defined by the TM $\mathcal{RL}$ R, is in AT-LEAST-LINEAR

However, $\forall x$ [Z(x) = Z'(x)], so they have the same I/O behavior and yet one is in and the other is out of AT-LEAST-LINEAR, denying the 2nd condition of Rice's Theorem

***b.*) HAS-IMPOSTER = { f | $\exists$ g [g≠f and $\forall y$ [$\varphi_g(y) = \varphi_f(y)$]] }.**

We can deny the 1st condition of Rice's Theorem since all functions have an imposter. To see this, consider, for any function f, the equivalent but distinct function g(x) = f(x) + 0. Thus, HAS-IMPOSTER is trivial since it is equal to $\aleph$, the set of all indices.

9. The trace language of a computational device like a Turing Machine is a language of the form
Trace = { C_1#C_2# … C_n# | $C_i \Rightarrow C_{i+1}$, $1 \le i < n$ }
Trace is Context Sensitive, non-Context Free. Actually, a trace language typically has every other configuration word reversed, but the concept is the same. Oddly, the complement of such a trace is Context Free. Explain what makes its complement a **CFL**. In other words, describe the characteristics of this complement and why these characteristics are amenable to a **CFG** description.

The complement of a trace need either not look like a trace (that's easy) or look like one, but have one or more errors. By one or more errors, we just mean that there is a pair C_j#C_{j+1}# where it is not the case that $C_j \Rightarrow C_{j+1}$. A PDA can guess which configuration starts this pair, push that configuration into its stack and check that the next one is in error (of course, this generally means one element of the pair is reversed). Such checking is within the capabilities of a PDA.

10. We described the proof that **3SAT** is polynomial reducible to Subset-Sum.
a.) Describe **Subset-Sum**
b.) Show that **Subset-Sum** is in **NP**
c.) Assuming a **3SAT** expression **(a + ~b + c) (~a + b + ~c)**, fill in the upper right part of the reduction from **3SAT** to **Subset-Sum**.

	a	**b**	**c**	**a + ~b + c**	**~a + b + ~c**
a	**1**			**1**	
~a	**1**				**1**
b		**1**			**1**
~b		**1**		**1**	
c			**1**	**1**	
~c			**1**		**1**
C1				**1**	
C1'				**1**	
C2					**1**
C2'					**1**
	1	**1**	**1**	**3**	**3**

11. Consider the decision problem asking if there is a coloring of a graph with at most k colors (k-Color), and the optimization version that asks what is the minimum coloring number of a graph (MinColor). You can reduce in both directions. So, do that. Make sure you carefully explain for each direction just what it is that you are proving.

1. Show that k-Color is polynomial time many-one reducible to MinColor:
Let G=(N,V) be an arbitrary graph and k>0 an arbitrary positive whole number. We can solve k-Color by asking an oracle for MinColor to provide the minimum coloring of G. We then check the MinColor answers. If it is less than or equal to k, we answer **yes**; else we answer **no**. This proves that MinColor is NP-Hard, based on our knowledge that k-Color is NP-Complete.

2. Show that MinColor is polynomial time Turing reducible to k-Color:
Let G=(N,V) be an arbitrary graph. Let n=|N|. Do a binary search suing an oracle for k-Color in order to find the minimum k such that k-Color returns **yes**. This takes log n questions of the oracle. Thus, MinColor is in FNP and is therefore NP-Hard. In fact, since we asked just log n questions and the reduction steps are polynomial, computing MinColor is no more complex than k-Color, within a polynomial factor.